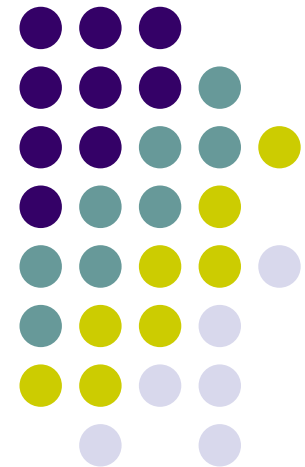


Einführung in die Systemprogrammierung

Grundlagen
Starten und Beenden von Prozessen



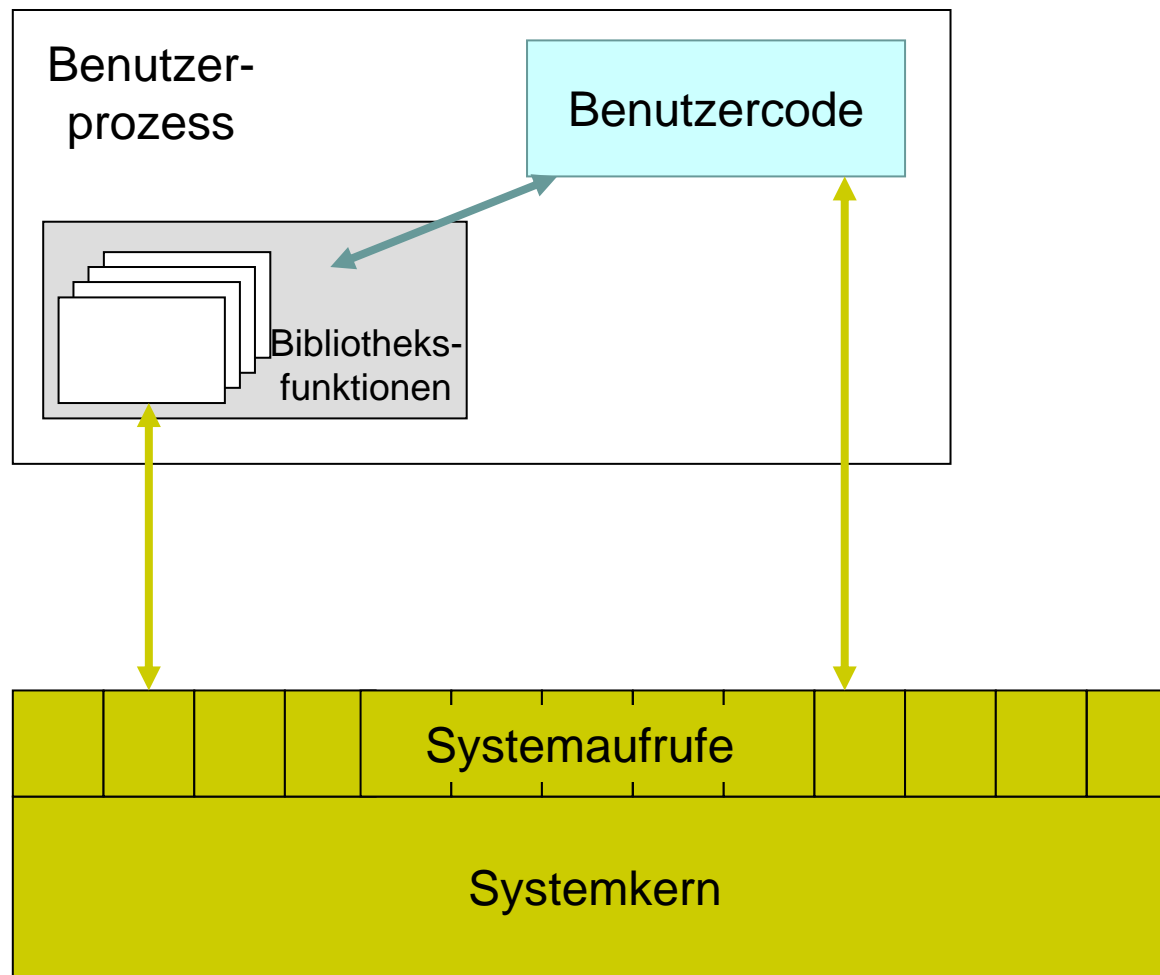
Systemaufrufe und Bibliotheksfunktionen

- Systemaufrufe
 - sind Systemkern-Schnittstellen
 - sind in Sektion 2 des „UNIX Programmer’s Manual“ beschrieben
 - Beim Aufruf wird Systemkern-Code ausgeführt
 - Befinden sich in der C-Standardbibliothek
 - Aus Benutzersicht kein Unterschied zwischen Systemfunktionen und Bibliotheksfunktionen
 - Austausch von Systemfunktionen erfordert Änderung des Kernels

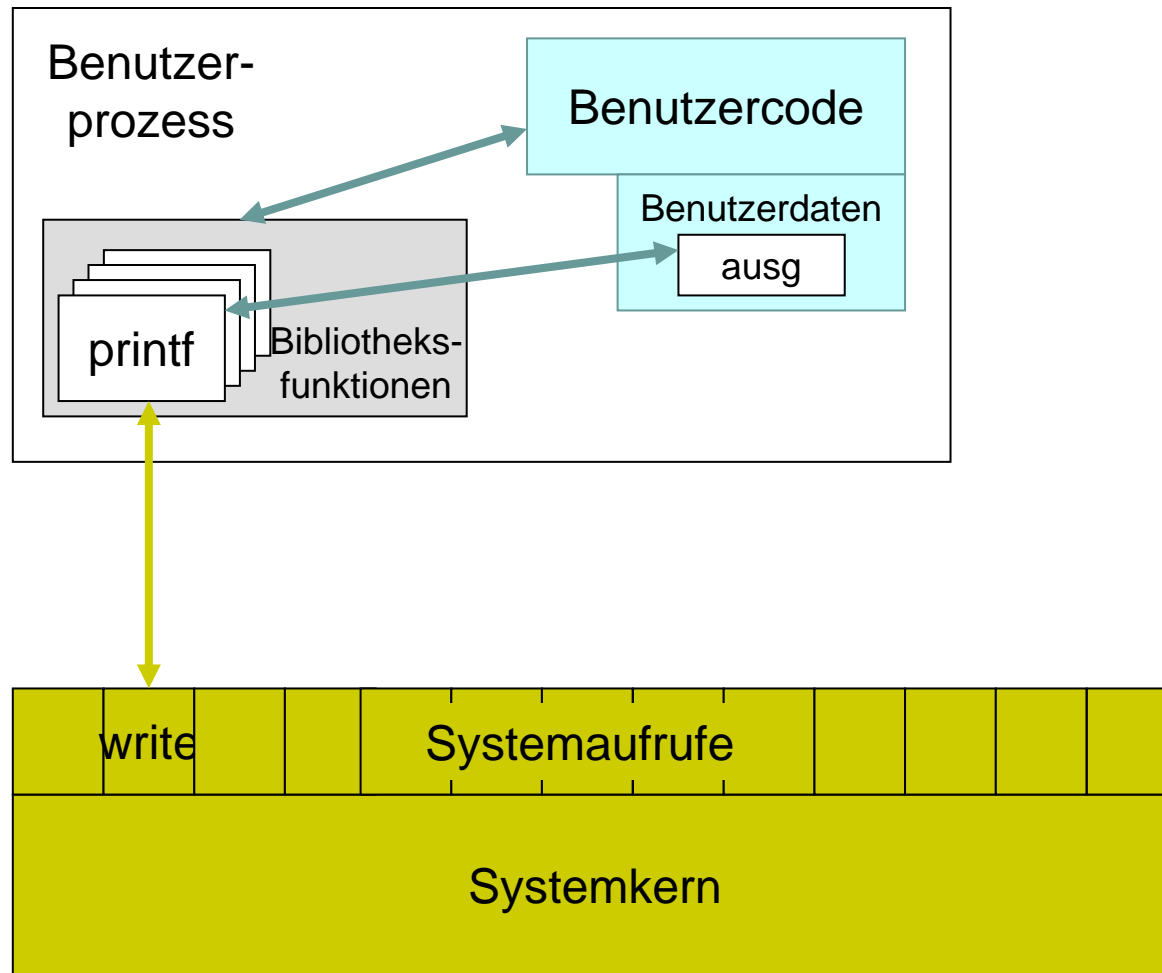
Systemaufrufe und Bibliotheksfunktionen

- Bibliotheksfunktionen
 - sind keine Schnittstellen zum Systemkern
 - sind in Sektion 3 des „UNIX Programmer’s Manual“ beschrieben
 - Führen größtenteils Benutzerfunktionen aus
 - können aber eine oder mehrere Systemfunktionen aufrufen
 - ***sqrt***, ***strlen*** rufen keine Systemfunktion auf
 - ***printf*** ruft Systemfunktion ***write*** auf.
 - Befinden sich in der C-Standardbibliothek Können leicht durch neue ersetzt werden.

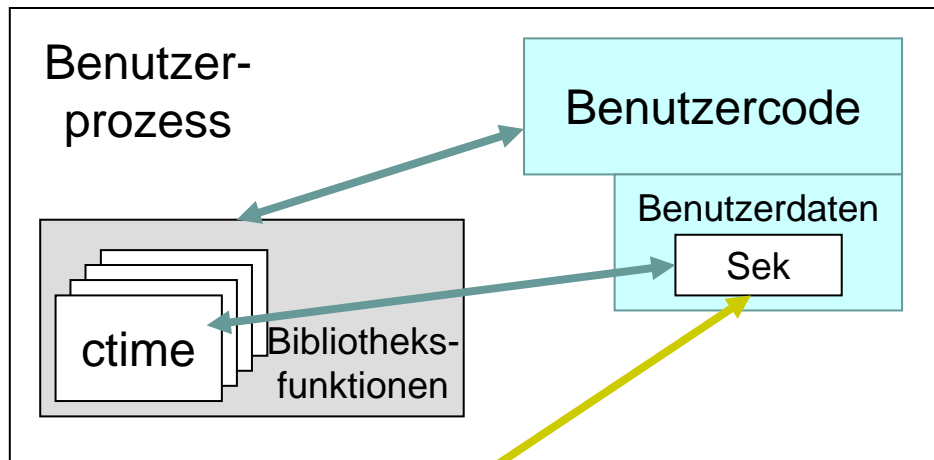
Systemaufrufe und Bibliotheksfunktionen



Beispiel 1: printf



Beispiel 2: time

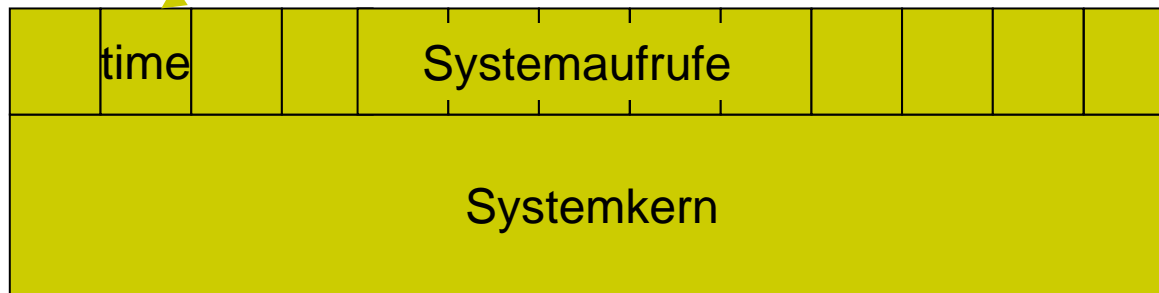


Systemaufruf time:

Liefert Julianisches Datum

Bibliotheksfunktion ctime:

konvertiert Jul. Datum in verständliches Datumsformat



C-Standardbibliothek-Headerdateien

Headerfile	Beschreibung
<code><assert.h></code>	Diagnostics
<code><ctype.h></code>	Character Class Tests
<code><errno.h></code>	Error Codes Reported by (Some) Library Functions
<code><float.h></code>	Implementation-defined Floating-Point Limits
<code><limits.h></code>	Implementation-defined Limits
<code><locale.h></code>	Locale-specific Information
<code><math.h></code>	Mathematical Functions
<code><setjmp.h></code>	Non-local Jumps
<code><signal.h></code>	Signals
<code><stdarg.h></code>	Variable Argument Lists
<code><stddef.h></code>	Definitions of General Use
<code><stdio.h></code>	Input and Output
<code><stdlib.h></code>	Utility functions
<code><string.h></code>	String functions
<code><time.h></code>	Time and Date functions

Referenz: <http://www.infosys.utas.edu.au/info/documentation/C/CStdLib.html>

Systemaufrufe: Header-Dateien

Headerfile	Beschreibung
<code><dirent.h></code>	Directory-Einträge
<code><fcntl.h></code>	Filekontrolle
<code><grp.h></code>	Gruppenfile
<code><pwd.h></code>	Passwortfile
<code><signal.h></code>	Signalhandling
<code><termios.h></code>	Terminalkontrolle
<code><unistd.h></code>	Symbolische Konstanten
<code><utime.h></code>	Filezeiten
<code><sys/stat.h></code>	Fileattribute
<code><sys/times.h></code>	Prozesszeiten
<code><sys/types.h></code>	Primitive Systemdatentypen
<code><sys/utsname.h></code>	Systemidentifikation
<code><sys/wait.h></code>	Prozesskontrolle



Primitive Systemdatentypen

Datentyp	Beschreibung
<code>clock_t</code>	Clock-Ticks
<code>dev_t</code>	Device-Nummer
<code>fd_set</code>	Filedeskriptormenge
<code>fpos_t</code>	Fileposition
<code>gid_t</code>	Gruppen-Identifizier
<code>ino_t</code>	Inode-Nummer
<code>mode_t</code>	Filetyp und -zugriffsrechte
<code>nlink_t</code>	Anzahl Links
<code>off_t</code>	Filegrösse und -offset
<code>pid_t</code>	Prozess-und Prozessgruppen-Identifizier
<code>sigset_t</code>	Signalmenge
<code>size_t</code>	Anzahl Bytes (vorzeichenlos)
<code>ssize_t</code>	Anzahl Bytes (mit Vorzeichen)
<code>time_t</code>	Anzahl Sekunden
<code>uid_t</code>	User-Identifizier
<code>wchar_t</code>	Zeichencodes

Der UNIX-Prozess

- Prozess=Programm in Ausführung
- Start eines Unix-Prozesses
 - Ausführung eines C-Programms beginnt immer mit der Funktion *main*.
 - Dieser Funktion ist eine Startup-Routine vorgelagert
 - wird vom Linker zum ausführbaren Programm dazugebunden
 - Ist eigentliche Startadresse des Programms
 - sorgt für den Aufruf der Funktion *main*.
 - Versorgt Prozess mit Kommandozeilenargumenten und Umgebungsvariablen

Übergabe von Befehlszeilenargumente

- *argc*: Anzahl der Argumente
- **argv[]*: Zeiger auf ein Array, das die Argumente enthält
- *argv[0]* ist der Programmname.

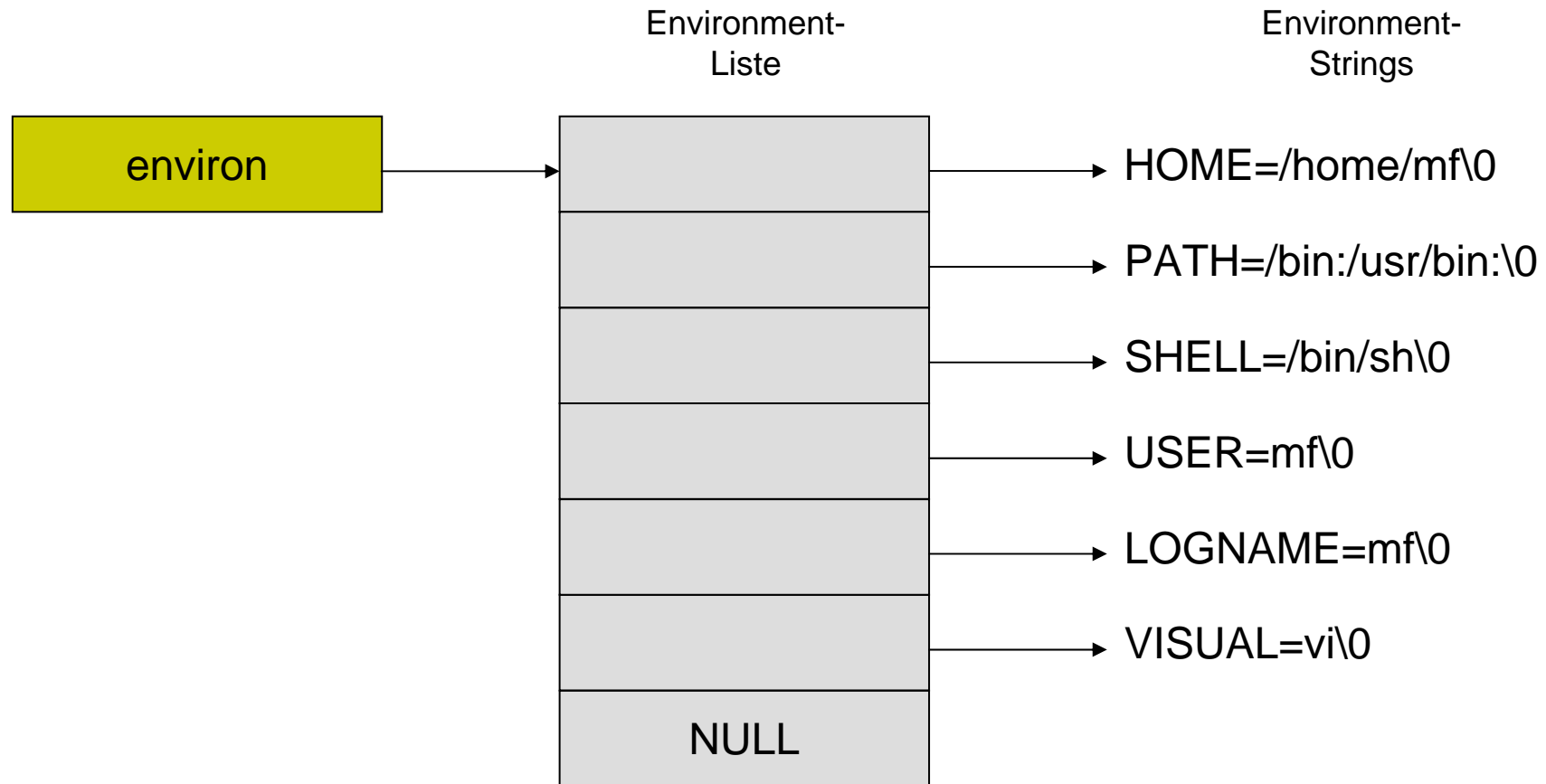
```
int main(int argc, char *argv[] )  
{  
  ...  
}
```

Umgebungsvariablen

- Jedem Programm wird über die globale Variable *environ* Liste von Umgebungsvariablen zugänglich gemacht
- *environ* ist ein Pointer auf einen Array von Pointern, die auf Strings der Form Name=Wert zeigen
- Letztes Element ist Nullpointer.

```
extern char **environ;
```

Umgebungsvariablen



Setzen/Lesen von Umgebungsvariablen

- getenv:
 - Erfragen des Wertes einer einzelnen Environment-Variablen
- putenv, setenv
 - Setzt Umgebungsvariablen
- unsetenv
 - löscht den Wert einer einzelnen Umgebungsvariablen
- cleanenv
 - löscht die gesamte Liste an Umgebungsvariablen

```
#include <stdlib.h>
char *getenv(const char *name);
int putenv (const char *eintrag);
int setenv (const char *name, const char *wert, int uebersch);
void unsetenv(const char *name);
```

Argumente und Umgebungsvariablen

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
extern char **environ;

int main( int argc, char *argv[] )
{
    int i;
    printf( "Befehlszeilenargumente:\n" );
    for(i=0;i< argc; i++ )
        printf( "%s\n", argv[i] );
    printf( "\nUmgebungsvariablen:\n" );
    for(i=0; environ[i] != NULL; i++ )
        printf( "%s\n", environ[i] );
    exit( 0 );
}
```

Umgebungsvariablen

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    char *s;

    // setzen einer Umgebungsvariablen
    putenv("DAU=doeddel");          // oder:
    setenv ("DAU", "doeddel", 1);

    // lesen einer bestimmten Umgebungsvariablen
    getenv("DAU");
    exit( 0 );
}
```

Beendigung eines UNIX-Prozesses

- Normale Beendigung
 - normales Beenden der Funktion *main* (mit oder ohne *return*)
 - Aufruf der Funktion *exit* oder *_exit*
 - Sinnvoll: Exitstatus!
 - *return(0);*
 - *exit(0);*
 - *_exit(0);*
- Abnormale Beendigung
 - Aufruf der Funktion *abort*
 - durch interne oder externe Signale

exit und _exit

- *exit*:
 - Programm beenden mit cleanup
 - Normale Programmbeendigung mit vorherigem
 - leeren der Puffer
 - schließen aller geöffneten Dateien
 - löschen aller temporär angelegten Dateien
 - *exit* ruft nach cleanup *_exit* auf.

```
#include<stdlib.h>
void exit (int status);
```

- *_exit*:
 - #include <unistd.h>
 - Programm normal beenden ohne vorherigen cleanup

```
#include<unistd.h>
void _exit (int status);
```

atexit – Einrichten von Exithandlern

- Registrierung von bis zu 32 Funktionen, die automatisch bei Beendigung des Programms mit *exit* aufgerufen werden.
- atexit trägt Funktion in Liste von Funktionen ein
- Funktionen werden in umgekehrter Reihenfolge der Registrierung am Programmende aufgerufen

```
#include <stdlib.h>

int atexit(void(*funktion) (void));
```

Beispiel atexit

```
#include<stdlib.h>
static void goodbye(void), tschuess(void);
int main(void)
{
    if (atexit(tschuess) !=0) //Registrierung tschuess
        printf("Fehler bei Registrierung von tschuess\n");
    if (atexit(goodbye) !=0) // Registrierung goodbye
        printf("Fehler bei Registrierung von goodbye\n");
    printf("Das wars schon!\n");
    exit(0); // oder auch return(0)
}
static void goodbye(void)
{
    printf("Goodbye\n");
}
static void tschuess(void)
{
    printf("tschuess\n");
}
```

Zusammenfassung

